@keithpitt in da house

MC. Keith
Fresh Prince of Builds

enjoy responsibly
I’m excited!
But first... history 🕒
Probably the Future
Glad I said “probably”
Probably the Future
Probably the Future
Is the future.
import React from 'react';

export default class UserAvatar extends React.Component {
  static propTypes: {
    name: React.PropTypes.string.isRequired,
    url: React.PropTypes.string.isRequired,
  }

  render() {
    return (
      <img src={this.props.url} className="UserAvatarComponent" alt={this.props.name} />
    );
  }
}
render() {
  return (
    <div className="BuildComment">
      <div className="BuildComment__avatar">
        <UserAvatar name={this.props.user.name} url={this.props.user.avatarURL} />
      </div>
      <div className="BuildComment__content">
        <BuildCommentTriangle/>
        <div className="BuildComment__content__body">{this.props.body}</div>
        <div className="BuildComment__content__footer">
          <div className="BuildComment__content__footer__name">{this.props.user.name}\</div>
          <div className="BuildComment__content__footer__status">{this.status()}</div>
        </div>
      </div>
    </div>
  );
}
Just the UI

Lots of people use React as the V in MVC. Since React makes no assumptions about the rest of your technology stack, it's easy to try it out on a small feature in an existing project.
Flux
how to get data into react

Load Initial Data via AJAX | React
https://facebook.github.io/react/tips/initial-ajax.html
Getting Started · Tutorial · Thinking in React ... Fetch data in componentDidMount.
When the response arrives, store the data in state, triggering a render to...

Displaying Data | React - Facebook Code | Facebook
https://facebook.github.io/react/docs/displaying-data.html
React makes it easy to display data and automatically keeps the interface ... focus on the JavaScript code and assume it's inserted into a template like the one above, ...
Babel exposes a number of ways to get started using JSX, ranging from ...

Tutorial | React - Facebook Code | Facebook
https://facebook.github.io/react/docs/tutorial.html
createClass(displayName: 'CommentBox', render: function()) { return (React ... We need to get this data into CommentList in a modular way. Modify ...

Thinking in React | React - Facebook Code | Facebook
Step 3: Identify the minimal (but complete) representation of UI state # To make your UI interactive, you need to be able to trigger changes to your underlying data model. React makes this easy with state. To build your app correctly, you first need to think of the minimal set of mutable state that your app needs.

loading json data from local file into React JS - Stack Overflow
Load Initial Data via AJAX | React
https://facebook.github.io/react/tips/initial-ajax.html
Getting Started · Tutorial · Thinking in React ... Fetch data in componentDidMount. When the response arrives, store the data in state, triggering a render to ...

Displaying Data | React - Facebook Code | Facebook
https://facebook.github.io/react/docs/displaying-data.html
React makes it easy to display data and automatically keeps the interface ... focus on the JavaScript code and assume it's inserted into a template like the one above. ... Babel exposes a number of ways to get started using JSX, ranging from ...
use strict

assign = Webpack.require("object-assign")

_agents = Immutable.Map()
_loaded = {}
_ruleMaps = {}

CHANGE_EVENT = 'change'

_addAgent = (agent) =>
  # Load the agent into the map
  _agents = _agents.set(agent.id, agent)

  # Invalidate the rule map cache
  _ruleMaps[agent.id] = null

_loadAgentsForAccount = (accountID) =>
  if (!_loaded[accountID]
    Buildbox.AgentAPI.getAllForAccount(accountID).done (agents) =>
      _addAgent(agent) for agent in agents
      _loaded[accountID] = true

      Buildbox.NewAgentStore.emitChange()
    null
  else
    _agents

normalizeQuery = (query) =>
for metaData in agent.metaData
    parts = metaData.split(/([^=]+)=([^=]+)\?/)  # Match key=value wildcards
    key = parts[1]
    value = parts[2]

    # Add the meta data value to the map
    ruleMap[metaData] = true

    # Add the wildcard version
    ruleMap["#{{key}}=*"{{}}] = true

ruleMap

_doesQueryMatchAgent = (query, agent) ->
    ruleMap = _ruleMaps[agent.id] || generateRuleMapForAgent(agent)

    for rule in query
        # If the rule doesn't exist within the agent's rule map, then it doesn't
        # match.
        return false if !ruleMap[rule]

        # If nothing _didn't_ match, then it does match!
        true

Buildbox.NewAgentStore = assign({}, EventEmitter.prototype, {
    MAX_AGENTS: 100
}
)

emitChange: ->
    @emit(CHANGE_EVENT)

addListener: (callback) ->
    @addListener(CHANGE_EVENT, callback)
else if agentA.name < agentB.name
  -1
else
  1

agents: results.toList().toJS()

counts:
  total: totalCount
  matched: matchCount
query: query
error: null

Buildbox.NewAgentStore.normalizedQuery = (query) =>
  _normalizeQuery(query)

Buildbox.NewAgentStore.dispatchToken = Buildbox.NewDispatcher.register (payload) =>
  switch payload.action.actionType
  when 'AGENT_UPDATED'
    if Buildbox.NewAgentStore.hasChangeListeners()
      payload.action.agentFetcher()
        .then (agent) ->
          _addAgent(agent)
          Buildbox.NewAgentStore.emitChange()
_(_(ツ)_/\_
Too many random “Restful” JSON endpoints 👎
The server and the client were coupled 👎
Inefficient and slow 👎
Too much code 👎
Lots of boiler plate 👎
Hard to get started 👎
Telstra 👎
Data Fetching for React Applications

Dan Schafer & Jing Chen

@dischafer @jingc

React.js Conference

React.js Conf 2015 - Data fetching for React applications at Facebook

Facebook Developers

61,952
{
  "user": {
    "builds": [
      {
        "pipeline": {
          "slug": "retry-tester"
        },
        "organization": {
          "slug": "acme-inc"
        },
        "message": "awesome",
        "number": 9,
        "comments": [
          {
            "body": "Awesome comment!",
            "createdAt": "2015-11-25T04:51:16Z",
            "user": {
              "name": "Keith Pitt",
              "avatarURL": "xxx"
            }
          },
          {
            "body": "Test 12",
            "createdAt": "2015-11-25T04:53:23Z",
            "user": {
              "name": "Keith Pitt",
              "avatarURL": "xxx"
            }
          }
        ]
      }
    ]
  }
}
{ "user": { "builds": [ "pipeline": { "slug" }, "organization": { "slug" }, "message", "number", "comments": [ "body", "createdAt", "user": { "name", "avatarURL" }, { "body", "createdAt", "user": { "name", "avatarURL" } } ] } ] }
1 { 
2   "user": { 
3     "builds": [ 
4       "pipeline": { 
5         "slug": "retry-tester" 
6       }, 
7       "organization": { 
8         "slug": "acme-inc" 
9       }, 
10      "message": "awesome", 
11      "number": 9, 
12      "comments": [ 
13        "body": "Awesome comment!", 
14        "createdAt": "2015-11-25T04:51:16Z", 
15        "user": { 
16            "name": "Keith Pitt", 
17            "avatarURL": "xxx"
18        }, 
19        { 
20            "body": "Test 12", 
21            "createdAt": "2015-11-25T04:53:23Z", 
22            "user": { 
23                "name": "Keith Pitt", 
24                "avatarURL": "xxx"
25            } 
26        ] 
27    ] 
28   } 
29 }
{
  "user": {
    "builds": [
      {
        "pipeline": {
          "slug"
        },
        "organization": {
          "slug"
        },
        "message",
        "number",
        "comments": [
          {
            "body",
            "createdAt",
            "user": {
              "name",
              "avatarURL"
            }
          },
          {
            "body",
            "createdAt",
            "user": {
              "name",
              "avatarURL"
            }
          }
        ]
      }
    ]
  }
}
{
  "user": {
    "builds": [
      "pipeline": {
        "slug":
      },
      "organization": {
        "slug":
      },
      "message",
      "number"
    ]
  }
}
{
  "user": {
    "builds": [
      "pipeline": {
        "name",
        "slug"
      },
      "organization": {
        "name",
        "slug"
      },
      "message",
      "number",
      "state"
    ]
  }
}
A specification 😊
It’s represents data how we think about it 👍
It looks like JSON 👍
Simple HTTP Post 👍
You control what data is returned 👍
Typed 👍
query IntrospectionQuery {
  _schema {
    queryType { name }
    mutationType { name }
    subscriptionType { name }
    types {
      ...FullType
    }
    directives {
      name
      description
      args {
        ...InputValue
      }
      onOperation
      onFragment
      onField
    }
  }
}

fragment FullType on __Type {
  kind
  name
  description
  fields(includeDeprecated: true) {
    name
    description
    args {
      ...InputValue
    }
    type {
      ...TypeRef
    }
    isDeprecated
    deprecationReason
  }
  inputFields {
    ...InputValue
  }
  interfaces {
    ...TypeRef
  }
  enumValues(includeDeprecated: true) {
    name
  }
}
'data': {
    '__schema': {
        'queryType': {
            'name': 'Query'
        },
        'mutationType': {
            'name': 'Mutation'
        },
        'subscriptionType': null,
        'types': [
            {
                'kind': 'OBJECT',
                'name': 'Query',
                'description': 'The query root for this schema',
                'fields': [
                    {
                        'name': 'viewer',
                        'description': null,
                        'args': [
                        ],
                        'type': {
                            'kind': 'OBJECT',
                            'name': 'Viewer',
                            'oOfType': null,
                            'isDeprecated': false,
                            'deprecationReason': null
                        },
                        'inputFields': null,
                        'interfaces': [
                        ],
                        'enumValues': null,
                        'possibleTypes': null
                    },
                    {
                        'kind': 'OBJECT',
                        'name': 'Viewer',
                        'description': 'Same description',
                        'fields': [
                            {
                                'name': 'user',
                                'description': null,
                                'args': [
                                ],
                                'type': {
                                    'kind': 'OBJECT',
                                    'name': 'User',
                                    'oOfType': null,
                                    'isDeprecated': false,
                                    'deprecationReason': null
                                },
                                'inputFields': null,
                                'interfaces': [
                                ],
                                'enumValues': null,
                                'possibleTypes': null
                            },
                            {
                                'name': 'role',
                                'description': null,
                                'args': [
                                ],
                                'type': {
                                    'kind': 'OBJECT',
                                    'name': 'Role',
                                    'oOfType': null,
                                    'isDeprecated': false,
                                    'deprecationReason': null
                                },
                                'inputFields': null,
                                'interfaces': [
                                ],
                                'enumValues': null,
                                'possibleTypes': null
                            }
                        ]
                    }
                ]
            }
        ]
    }
}

```
IDE integration 😊
Before commit hook validation 👍
Ruby implementation of Facebook's GraphQL

- 396 commits
- 4 branches
- 23 releases
- 12 contributors

Latest commit 22e9323 7 hours ago

- guides: docs(executing_queries) fix typo in example query
- lib: 0.10.4
- spec: refactor(StaticValidation) use alias for error message
- .gignore: 0.7.1
- .travis.yml: Enable Travis CI cache
- .yardopts: doc(+) add lots of YARD docs
- CHANGELOG.md: 0.10.3
- Gemfile: syntax errors
- Display a menu: better field introspection
BuildType = GraphQL::ObjectType.define do
  name "Build"
  description "A build from a pipeline"
  field :id, !types.ID, "The unique ID for this build" do
    resolve -> (build, arguments, context) do
      build.uuid
    end
  end
  field :number, !types.Int, "The number of the build"
  field :message, !types.String, "The message for the build"
  field :commit, !types.String, "The commit for the build"
  field :branch, !types.String, "The branch for the build"
  field :organization, -> { OrganizationType } do
    resolve -> (build, arguments, context) {
      build.account
    }
  end
  field :pipeline, -> { PipelineType } do
    resolve -> (build, arguments, context) {
      build.project
    }
  end
  field :jobs, -> { JobConnectionType } do
    resolve -> (build, arguments, context) {
      Connection.create_from_relation(arguments, build.jobs)
    }
  end
end
UserType = GraphQL::ObjectType.define do
  name "User"
  description "A user"

  field :id, !types.ID, "The unique ID for this user" do
    resolve -> (user, arguments, context) do
      user.uuid
    end
  end

  field :name, !types.String, "The name of the user"
  field :avatarURL, !types.String, "The URL of the user's avatar" do
    resolve -> (user, arguments, context) do
      User::Avatar.url(user)
    end
  end
end
The glue between GraphQL and React
class BuildCommentsList extends React.Component {
    _handleSubmit = (e) => {
        // Boring code here
    }

    render() {
        let comments = this.props.comments
        let user = this.props.viewer.user

        return (
            <div className="BuildCommentsList">
                <div className="BuildCommentsList__comments">
                    {comments.map((c) =>
                        <div className="BuildCommentsList__comments_comment" key={c.id}>
                            <BuildComment state='posted' {...c} />
                        </div>
                    )
                </div>
                <div className="BuildCommentsList__form">
                    <BuildCommentForm onSubmit={this._handleSubmit} user={user} ref="commentForm" />
                </div>
            </div>
        );
    }
}
export default Relay.createContainer(BuildCommentsList, {
  initialVariables: {
    organization: null,
    pipeline: null,
    number: null
  },

  fragments: {
    viewer: () => Relay.QL`
      fragment on Viewer {
        user {
          name,
          avatarURL
        }
      }
    build(organization: $organization, pipeline: $pipeline, number: $number) {
      id,
      comments {
        nodes {
          id,
          body,
          user {
            name,
            avatarURL
          },
          createdAt
        }
      }
    }
  }
});
class BuildCommentsList extends React.Component {
  _handleSubmit = (e) => {
    // Boring code here
  }

  render() {
    let comments = this.props.viewer.build.comments.edges
    let user = this.props.viewer.user

    return (
      <div className="BuildCommentsList">
        <div className="BuildCommentsList__comments">
          {comments.map((c) =>
            <div className="BuildCommentsList__comments_comment" key={c.node.id}>
              <BuildComment state='posted' {...c.node} />
            </div>
          )}
        </div>
        <div className="BuildCommentsList__form">
          <BuildCommentForm onSubmit={this._handleSubmit} user={user} ref="commentForm" />
        </div>
      </div>
    );
  }
}
Batched requests 👍
Partial data fetching 👍
Telstra 👍
Things I skipped:
Things I skipped:

Webpack + Rails
Mutations
Security + Authorisation
Routes
Subscriptions
ES6,7,8,9,10,xx
Probably more…
Get excited
Is the future.
Buildkite GraphQL Explorer Beta

Enter a Buildkite API Token to get started:

[Input Field]

Login
# Welcome to the Buildkite GraphQL Explorer

# This is an in-browser IDE for writing, validating, and testing GraphQL queries.

# Type queries into this side of the screen, and you will see intelligent typeahead aware of the current GraphQL type system. Live syntax and validation errors highlighted within the text editor.

# To bring up the auto-complete at any point, just press Ctrl + Space.

# Press the run button above, or Cmd-Enter to execute the query. The result will appear in the pane to the right.

```
query AwesomeQuery {
  viewer {
    user {
      name
    }
  }
}
```
justin  3:04 PM

OOOOOOOOOOOH

oh wow
justin  3:06 PM
fantastic
dude, that's amaze
justin  3:09 PM
far out

ok

yeah

keithpitt  3:10 PM
see what I mean?

justin  3:10 PM
composing queries this way is amazing
incredible
justin  3:25 PM

ahh, cute
	nice
justin 3:26 PM
geking out hard

😊
**justin**  3:26 PM
yeah awesome

I know how it works in theory
but actually using it is another thing

😊

hadn’t appreciated how powerful it all is
thanks for the preview!

excited!

**keithpitt**  3:28 PM
no worries!
i’m excited that your excited

**justin**  3:29 PM
haha

😊
CURL -X POST \
-H "Authorization: Bearer xxx" \
https://graphql.buildkite.com/v1 \
-d '{ "query": "query AwesomeQuery { viewer { user { name } } }" }'
Fin ❤️

@keithpitt